Project 3: Algorithm Animation using Galant

- This project must be submitted by 10:00 PM, on *the date specified in the course schedule or via a Piazza note (the latter takes precedence).*
- Please submit a file called *uid*`_project_3.zip`, where *uid* is the unity id of the team leader, an archive containing all `.alg` files that implement your solution.[1] and a document describing your experience implementing these animations as described in more detail under **what to submit**.
- This project will count as 15% of your final grade.

Teamwork

As with previous project assignments, you are required to work in teams of 3 or 4 members unless you have obtained special permission from the instructor. Failure to follow this instruction will mean *no credit for the project.*

By 10:00 PM, April 15, you are required to register any *changes in team composition.* To do so, post a message with the names and unity id's of all team members and the team name of a new team on Piazza (you can reuse the name of an existing team if it's a matter of, for example, replacing a single member). If an existing team is disbanding, post an message with that information. All team composition posts should be posted to the `teams` folder. If your team is staying together, there is no need to register again. I prefer that you stay together, but, if that's not desirable, I need to know why.

When working as a team on code and documentation, consider reviewing each other's code instead of merely assigning a different piece for each team member to create independently. I also strongly recommend pair programming – let me know if you're unfamiliar with this practice.

Academic Integrity

You ***are*** permitted to discuss your general approach, algorithms, and results with people who are not on your team – Piazza is an excellent forum for doing this. However, ***no*** actual code or documentation can be shared. This prohibition applies to code produced by another person in or for previous offerings of this course or another course. Any such sharing will be considered an academic integrity violation. You are welcome to use code in the textbook or another source, but using such code without citing your sources constitutes plagiarism. ***Cite your sources in comments.*** Do this for any classes/methods you didn't write entirely yourself (and don't modify the borrowed code – you need to practice good code reuse). You are also allowed to use classes and methods in the Java API if appropriate. In case of doubt about what is permitted, ask the instructor or check the Code of Student Conduct (see syllabus for the link)

Learning outcomes

Upon completion of this project you will be able to

1. demonstrate understanding of graph algorithms
2. demonstrate understanding of recursive sorting algorithms
3. create compelling visualizations of algorithms

[1]The `.alg` extension is used by Galant for files that contain implementations of algorithm animations.

4. navigate documentation for an unfamiliar tool that relies on your understanding of graph representations and graph algorithms
5. write about the effectiveness of algorithm animation as an educational tool and the challenges involved in creating effective animations

Overview

Galant, an acronym for Graph algorithm animation tool,[2] is the name of a recently developed piece of software that simplifies the process of creating compelling graph algorithm animations.

A host of algorithm animation programs have been developed over the years. Primarily these have been designed for classroom use and involve considerable overhead for the creator of the animations (an instructor or developer) — students are passive observers. Three primary roles are characteristic of algorithm animation: the *observer*, who simply watches an animation; the *explorer*, who is able to manipulate problem instances(e.g., graphs); and the *animator*, who designs an animation. A key feature of Galant is that it simplifies the role of the animator, enabling users to create their own animations by adding a few visualization directives to implementations of algorithms in Java style pseudocode. The focus on graph algorithms has two key advantages: (i) the objects manipulated in an animation all have the same type; and (ii) graphs are ubiquitous and therefore provide a framework for animations in domains beyond classic graph algorithms. Examples include search trees, automata, and even sorting.

Galant is also distinguished in that it is a tool rather than a closed system. In other words, it is designed to interact easily with other software such as text editors, other graph editors, other algorithm animation tools, graph generators, Java API's, format translation filters and graph drawing programs. This interactivity significantly expands the range of Galant's applications, including, for example, as a research tool for exploring graph algorithms.

Expectations

Each team is expected to produce three algorithm animations. For one of these, choose between animating Boruvka's minimum spanning tree algorithm or the biconnected components algorithm described in a separate document, `bicon.pdf`. For the second, choose between merge sort, Quicksort, or Heapsort (these will require some creativity). For the third choose an additional algorithm from among the five listed so far or another algorithm of your choice, with my permission (there are lots of graph algorithms that are relatively easy to understand and implement). Also expected is a document describing various aspects of your experience using Galant – see below under **what to submit** for more details.

Files provided

The following files are posted on the Moodle site along with this project description.

- `Galant.jar` – this is the executable code for Galant.
- `2014_galant_tr.pdf` – a technical report describing all aspects of the purpose, design, and use of Galant, including documentation for explorers (user documentation) and animators (programmer documentation), bugs, and inconveniences.
- `bicon.pdf` – a pseudocode description of the biconnected components algorithm, in case you decide to implement an animation of it.

[2]Aside from being an acronym, Galant is a term for a musical style that featured a return to classical simplicity after the complexity of the late Baroque era. Galant hopes to achieve the same in the context of algorithm animation.

- `Galant-104.zip` – the source code for Galant in case you really want to know "how sausage is made". [3] The actual code lives in a tree of subdirectories under `src/edu/ncsu/csc/Galant`. You can execute Galant from the top level by issuing the `ant` command.
- `animations.zip` – contains animations (`.alg` files) of eight algorithms plus four files containing animation code for testing purposes.
- `graphs.zip` – contains five graphs (`.graphml` files) that can be used to test various algorithms: `eight_node_graph` is for testing depth-first or breadth-first search, `sssp_505_example` and `weighted_example` are for weighted graph algorithms, `sorting_test` is for sorting algorithms, and `test`, used to try out different node and edge attributes. The sorting test consists of four nodes randomly scattered with weights that determine their desired sorted order. The bubble sort and insertion sort animations begin by putting the nodes in a straight line.
- Other files may be provided as needed.

Implementation notes

To run Galant, simply execute the jar file from the command line –

```
java -jar Galant.jar
```

or double-clicking on it (works on my Mac, probably in Windows, too). This causes two windows to appear as described in the technical report – please read this before continuing, lest you ask unnecessary questions. You should set the default directory that the file browser will use when you want to open or save a file: File → Preferences → Open/Save. The File → Open menu option (text window) allows you to open either a graph or an algorithm.

I strongly recommend that you begin by running some of the existing animations (including the three test ones) and modifying them slightly to get a feel for how Galant works.

What to submit

You should submit a `zip` archive – *no `tar` or `rar` archives, please* – containing the following.

1. Three `.alg` files implementing your three animations; you may also include some `.java` files implementing supporting classes; animations can be designed to import these;

2. A pdf document describing your work; the document should have the following sections.
   - An overview of each animation with two or three screen shots illustrating its execution. The reader should be able to run and understand the animation from your description.
   - A list of comments about the process of creating the animations. There should be a bulleted or numbered list of items in each of the following categories:
     - comments about the features of Galant that made it easy to create the animations;
     - comments about Galant's shortcomings and bugs not listed anywhere in the documentation;
     - other comments.
   - A paragraph or two answering the questions: "What did I learn from working on these animations?" and "Did I gain a better understanding of the algorithms by animating them instead of working through them on examples or watching examples presented in class?"

In addition, each of you individually must fill out evaluations of your teammates on a Google form provided for that purpose, as you did with the second project.

Before you get credit for this project and a grade for CSC 316, you must provide me with a completed and signed consent form. A blank form is posted on the Moodle site with the other project-related

[3] A team of undergraduate research assistants will implement a variety of improvements this summer.

files. The form gives you the option of opting in or out – either way you need to complete and sign it to get credit for this project. For convenience submit the scanned form to the `consent_form` Wolfware Classic submit locker, or, if you prefer, give me a hard copy. If you *opt in*, the form allows me to use your animation(s) in future classes or use excerpts from the document you submit in some future technical report or paper about Galant. It also applies to other projects, specifically Project 1, should I want to use your charts, tables, or excerpts of your explanations as examples to emulate.

Grading

The table below gives a general idea of how points will be assigned. I say "general idea" because boundaries between some of the categories may be blurred, such as "clearly illustrates the algorithm" and "makes effective use of Galant functionality".

category		**points**
Each animation . . .		30
compiles and executes	5	
clearly illustrates the algorithm	10	
makes effective use of Galant functionality	5	
is described and illustrated effectively in the document	10	
Total for animations		90
The document has comments about Galant features and shortcomings (several of each, described clearly, using examples as appropriate)		10
The document gives well-written answers to the two questions described above		10
Other team members give positive evaluations		10
Total		**120**

100 points is a perfect score. The remaining 20 points are "built-in" extra credit.